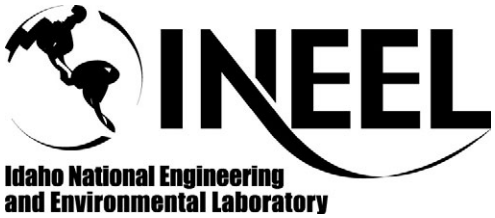

INEEL/CON-04-02012
PREPRINT

Visualization Of A Deterministic Radiation Transport Model Using Standard Visualization Tools

James A. Galbraith
L. Eric Greenwade

May 17 – 24, 2004

Cray User Group Meeting

Visualization Of A Deterministic Radiation Transport Model Using Standard Visualization Tools

James A. Galbraith and **L. Eric Greenwade**, Idaho National Engineering and Environmental Laboratory

ABSTRACT: *Output from a deterministic radiation transport code running on a CRAY SV1 is imported into a standard distributed, parallel, visualization tool for analysis. Standard output files, consisting of tetrahedral meshes, are imported to the visualization tool through the creation of a application specific plug-in module. Visualization samples are included, providing visualization of steady state results. Different plot types and operators are utilized to enhance the analysis and assist in reporting the results of the analysis.*

1 Introduction

As technological advances continually increase our ability to acquire and create scientific data, it becomes increasingly important to develop techniques to analyze and aid in the understanding of this complex scientific data. These advances allow our scientists and engineers to study in more detail much larger dynamics through the use of more realistic simulations. These more realistic simulations result in the generation of increasing amounts of data.

For scientific applications – especially those associated with high-performance numerical computing – some of the most widely encountered object types are meshes, and fields based on those meshes. A mesh represents a discretization of a space, a decomposition into many thousands, or in some cases millions, of smaller-sized objects known as cells. Fields are defined by associating values with the cells.

While we could create a specific solution to a specific problem, our need is to find a solution to a more generic problem. Our need is to find and integrate a generic visual data analysis solution to many problems found at the INEEL.

This paper presents the activities associated with the visual data analysis of one of the modeling codes in use at the Idaho National Engineering and Environmental Laboratory (INEEL), named Attila [1][2]. While this paper focuses on available visual data analysis techniques and how they have been applied to Attila and it's results, the techniques and capabilities of Attila itself are presented at a high level for background information only.

2 Data Analysis

The INEEL has a diverse pool of scientific and engineering tools that result in the creation of scientific data. These tools are used by a diverse range of disciplines, including: nuclear safety analysis, reactor core design, subsurface science, among others. The size of the problem also varies widely across the different disciplines.

Larger models may be run on one of our Cray SV-1 platforms or an Opteron based cluster. Smaller models may be run on various shared memory multiprocessor (SMP) platforms such as available from Sun or SGI. Even smaller models may be run on desktop SMP platforms or even single processor workstations.

Regardless of the compute platform the model is executed, the results are typically stored in one or more disk file(s). These disk files may be in either standard ASCII or machine specific binary formats. ASCII files may be examined manually, which becomes a daunting task as the resulting data files become larger. Binary files require the use of analysis tools to read the binary data and provide analysis output in a form desired by the user. ASCII files may also me analyzed using the same types of tools. Knowledge of data formats is required for analysis tools to utilize either the binary or ASCII data formats as input.

Manual examination of ASCII results data allows the user to examine data in extreme detail. But it is possible for the user to quickly loose track of the big picture. One might easily examine data from a single time step or a single or group of cells, but is unable to easily correlate that data against previous or following timesteps, or even against the overall cell structure. Extraction of portions of the results data into spreadsheet applications for processing or two-dimensional plotting packages may provide a better view of the data, but a complete overall view of the results is still not available.

Analysis of binary data through software utilities also can lead to a loss of the big picture depending on the method the results of the analysis are presented.

In order to provide an adequate overall view of the results, some method of providing an interactive three-dimensional (3D) view of the data is required. At the same time, this 3D view must be able to animate over a series of timesteps where necessary.

The basic problem, simply stated, is to provide a visual data analysis tool to assist in the analysis of scientific data that is able to satisfy the various requirements of multiple disciplines. By providing a single or small set of visualization tool(s), we are then able to benefit from the use of shared resources, while minimizing custom software development, support, and training.

With each discipline having their own analysis techniques and potential data formats, attempting to apply generalized visualization techniques across each of these disciplines is a difficult task, especially for those of us not intimately involved with each discipline. We also need to address not only those problems that exist today, but also anticipate those problems that require visualization tomorrow, next year, and even further down the road.

Most users of existing modeling applications want visualization capabilities yesterday and are unable or unwilling to wait for new capabilities to be created, either integrated or stand-alone. There are also reasons why integrated visualization capabilities cannot or should not be developed for individual applications. First of all is the funding issue and time required to develop specific integrated capabilities. Other issues such as software architecture and the inability to easily integrate such features without severe re-engineering also prevail.

3 Visualization

It is important to differentiate activities associated with visual data analysis of scientific data and presentation graphics. Presentation graphics is primarily concerned with the communication of information and results in ways that are easily understood. In visual data analysis, we seek to understand the data. Presentation graphics may be used to present the results of the visual data analysis but are not necessarily involved in the analysis itself.

Visual data analysis is part of a much larger domain typically called visualization. Many of the tools used in visual data analysis are applied to other general areas of visualization.

Visualization as a whole is an emerging science, with fundamental ideas being applied to general tools with application across multiple disciplines. At times, it may be desirable to obtain and visualize scientific data on a real-time basis, but visual data analysis is typically done on historical data or data that has been produced by a simulation and saved for later use.

There are several advantages of 3D visualization of scientific data. First of all, large amounts of data can be combined for a single picture. All cells of a model can be combined with a single scalar value representing a specific state of the model at a specific time. This type of display can easily expose correlations within the data. Once the visualization has been displayed, it can then be interactively manipulated to allow the user to filter through the data to focus on specific portions if necessary. These displays can then be sequenced through many time steps of a dynamic model to expose correlations of data across multiple time steps.

Another advantage is the ability to not only visualize scalar data but also vector data, allowing magnitude or direction components of vector data to also be shown.

3.1 Techniques

Several different techniques are available for 3D visualization. Each of these techniques utilize color coding schemes to indicate data values either at the nodes or within the cells of the associated mesh. The following techniques are typically used for 3D visualization of scientific data:

- Isosurfaces
- Volumetric rendering
- Contours
- Animation

3.1.1 Isosurfaces

An Isosurface is a technique where the surface displayed is a constant scalar data value across the entire data domain. This results in a 3D contour display.

3.1.2 Volumetric rendering

Volumetric rendering is the opposite of surface rendering, where the entire model is visualized and one is able to see inside the model volume. This technique is commonly use with slices and clipping to remove portions of the volume to allow viewing of internal cells.

3.1.3 Contours

The technique of using contours is essentially providing a 2D representation of a slice of data from the 3D data set. A contour is simply a slice of an isosurface.

3.1.4 Animations

Animations are used to show correlations of data values across multiple data sets. It sometimes is simply not enough to view the correlations of data between individual cells within a single data set. It may be just as important to know what the value was before and after any single timestep and to view the data values as they change over time.

3.2 Tools

There are many visualization tools currently available in both the commercial (e.g. Tecplot [6], AVS [3]) and freeware (e.g. VisIt [7], ParaView [4]) markets. There are also lower level visualization toolkits (e.g. VTK [8]) available to assist in the creation of visualization solutions. Each of these tools and toolkits have their individual advantages and disadvantages.

While searching for the appropriate visualization solution for use here at the INEEL, one of the primary requirements is the support for extremely large datasets. Other specific needs include the need for extensibility and availability of various compute platforms.

3.3 *Requirements*

As mentioned earlier, the general requirements for selecting a visualization tool were gathered from potential users currently at the INEEL as well as from anticipated problems. Specific requirements included:

- Availability on the necessary compute platforms
- Support for existing data formats
- Support for extremely large data sets
- Visualization extensibility
- Input data format extensibility
- Ability to create "movies"
- 2D/3D support

Without going into a long description of our selection process, suffice it to say VisIt, from Lawrence Livermore National Laboratory (LLNL), was selected as our primary visualization of choice. It provides support for tera-scale data sets, is available in source distributions and portable across a multitude of platforms, utilizes plug-in capabilities to extend plot, operator, and data formats, provides the ability to create standard format output graphic files and animated movies, and provides an extensive set of standard visualization techniques.

4 VisIt

VisIt is a free, component based, interactive parallel visualization and graphical analysis tool developed at LLNL for viewing scientific data on Unix and PC platforms. The primary driving force behind the design and development of VisIt was for visualizing tera-scale data. While this is true, it is equally well suited for visualizing data from much smaller data sets typically encountered on desktop systems.

The most recent version of VisIt available to the public is 3.0. Releases are made periodically, typically every two to three months. Visit www.llnl.gov/visit for software distribution and documentation.

VisIt was designed to leverage visualization of others and component based to enhance it's extensibility. VisIt is based on the Visualization Toolkit (VTK) as provided by Kitware. In addition to this, new plot types, operators, and databases can be defined and implemented as a plugin using the appropriate shared library technology of the target platforms.

VisIt allows the user to have multiple displays open , allowing the user to simultaneously view the same data using different plot types and/or operators. Plots may be opened and closed as necessary. Operators are easily added to individual plots to filter data as required.

Interaction between the user and a plot is simple and intuitive. The user is able to easily manipulate the viewing location through standard pan, rotate, and zoom capabilities allow viewing the plot from the appropriate angle.

While VisIt provides a rich set of plot types, operators, and supported data formats, one of the more intriguing concepts of VisIt is the fact that it is component based. By providing and supporting plug-in technology for plots, operators, and data formats, the tool becomes incredibly extensible, only limited by the demands placed upon the tool and the imagination of the engineers solving the problem.

Before I describe some of the available plot, operator, and database options, please note that each of these plot types and operators has a variety of configurable elements. These elements range from line types, colors, opacity, scale values, and other elements that are specific to individual plot or operator types. Samples of the plot types and operators of interest to Attila are presented later in Section 6.

4.1 *Plot Types*

VisIt provides an extensive set of plot types for visualizing data, with each using different techniques to satisfy various needs. The different plot types provided, include: Boundary, Contour, Curve, Mesh, Pseudocolor, Streamline, Surface, Vector, and Volume. Plot types of primary interest of Attila include Contour, Mesh, Pseudocolor, Vector, and a combination Line-out plots.

The Contour plot type provides an isosurface display of the selected. The contour plot displays the location of values for scalar variables like density or pressure using lines for 2D plots and surfaces for 3D plots. Figure 1 presents a sample of the contour plot.

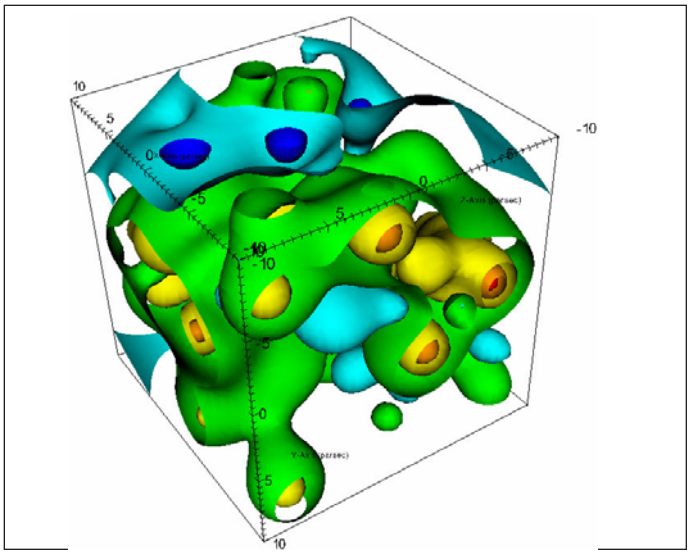

Figure 1 Contour (Isosurface) Plot

The Mesh plot type provides a view of the computational mesh associated with the current model. Mesh plots are typically displayed as a wireframe and utilized in conjunction with other plots to provide delineation between cells. Several different types of meshes are supported, including two- and three-dimensional rectilinear, curvilinear, and unstructured meshes.

Pseudocolor plots map scalar variable data values to colors and uses the colors to "paint" values onto the variable's compu-

tational mesh. The result is a clear picture of the database geometry painted with variable values that have been mapped to colors. Figure 2 presents a pseudocolor plot in conjunction with a mesh plot.

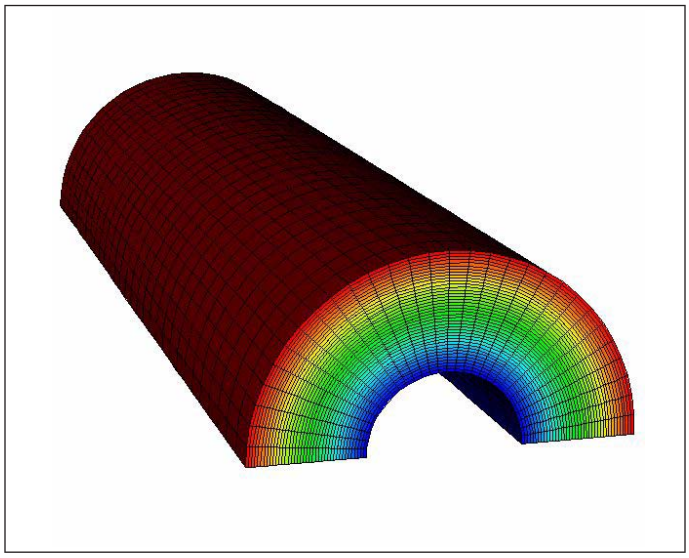

Figure 2 Pseudocolor Plot (with Mesh Plot)

Vector plots displays vector variables as small glyphs that indicate the direction and magnitude of a vector field. Figure 3 presents a simple vector plot in conjunction with a mesh plot.

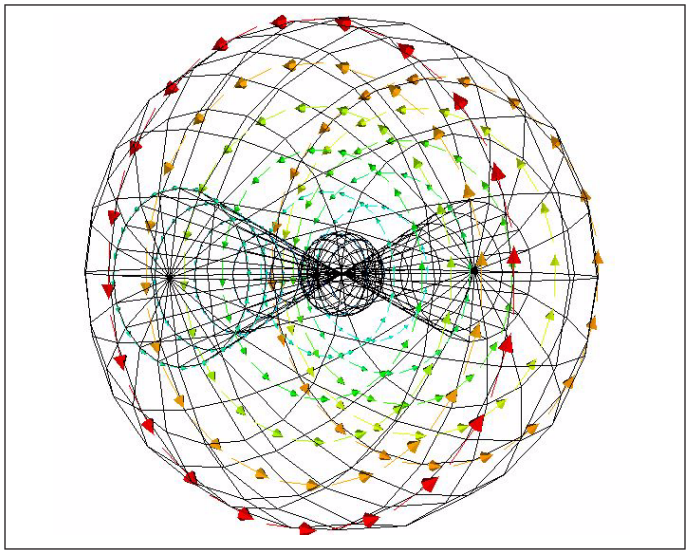

Figure 3 Vector Plot (with Mesh Plot)

4.2 Operators

Operators are filters that can be applied to VisIt database variable before a plot is generated. Several operators are provided with the VisIt distribution that provide data restriction operations such as planar and spherical slicing, and thresholding. Other more sophisticated operators are also available. Operators of primary concern to Attila are the Clip, Slice, and Lineout operators.

The Clip operator removes portions of the VisIt database before the plot is generated. Shapes removed are typically defined with planes and spheres. Multiple planes can be defined to allow multiple clipped surfaces. Figure 4 presents a simple spherical plot clipped by two planes.

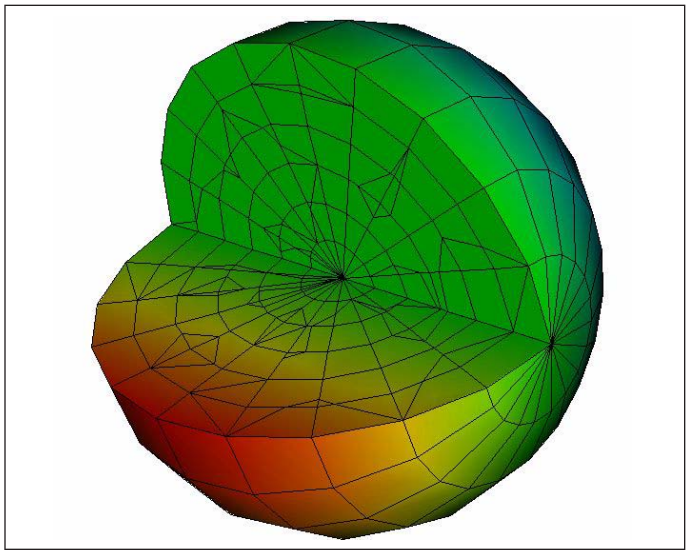

Figure 4 Clip Operator

The Slice operator slices a 3D database with a plane with an arbitrary orientation. Plots to which the Slice operator has been applied are turned into 2D planar surfaces that are coplanar with the slice plane. The resulting plot can be left as a 2D slice in 3D space or it can be projected to 2D space where other operations can be done to it.

The Translate operator allows the user to manipulate a plot using scale, rotation, and translation transformations.

4.3 Database Plugins

VisIt provides support for several standard data formats, including: Silo, SAF, VTK, and others. By implementing plugin technology for the input data streams, VisIt can easily be extended to support any data format by simply creating a database plugin module for the desired format.

4.4 Other Features

The lineout mode allows the user to draw a reference line in a 2D plot. VisIt will then extract data points along the reference and display the results in an Curve plot in another window.

The VisIt database can be divided into domains, where individual domains can be disabled and removed from view. In this manner the user can easily remove unnecessary portions of the database without the need to utilize complex operators. Due to the size of many of the Attila models, this feature is invaluable in allowing the viewer to easily display only the portions of the database currently of interest.

VisIt also supports writing images of a plot view to any of several currently supported file formats, including: tiff, jpeg, bmp, and png, among others. This allows the user to easily include visual representations of the model in reports and

presentations. These visualizations may be easily annotated through the annotation controls provided.

Many of these output files can also be used in the creation of animations or movies that can also be used for presentations and demonstrations. VisIt does not specifically support the creation of these movies other than through the ability to create the necessary snapshot images required by various movie making utilities.

VisIt has also implemented an interface to the Python scripting language to allow automated operation of certain capabilities. The Python interface is typically used to consistently create specific visualization environments images used in the creation of movies or other analysis and reporting activities.

5 Attila

The Attila© application is a deterministic radiation transport software package designed to solve the first order form of the Boltzmann transport equation on three dimensional unstructured tetrahedral meshes. In addition to providing neutron transport calculations, Attila is also able to perform charged particle transport calculations as well as perform infrared steady-state calculations for radiative transfer purposes.

Attila was designed and implemented to be a robust, general purpose radiation transport solver by the Computer Research and Applications Group at the Los Alamo National Laboratory (LANL) [1] and licensed to Radion Technologies [2] for commercial development. While applicable to a wide variety of radiation transport applications, specific applications include:

- Nuclear reactor design and analysis
- Radiation shielding and protection
- Medical therapy and imaging
- Charged particle calculations
- Food and equipment sterilization operations

Attila is implemented primarily in Fortran 95 using C preprocessor commands and consists of approximately 45000 lines. While the primary compute platform targeted by Attila is the uniprocessor desktop platform, due to the need to support larger problems at the INEEL, it has been successfully ported to the Cray environment for our larger problems with successful results.

Attila is currently under consideration for use at the Advanced Test Reactor (ATR) at the Idaho National Engineering and Environmental Laboratory (INEEL) in conjunction with core safety analysis activities.

The ATR is capable of creating a wide range of reactor environments in which the effects of radiation on materials and fuels may be studied. These tests determine how fuels and materials react when bombarded by streams of neutrons and gamma rays under a variety of pressure and temperature conditions. Information that would normally require years to gather from normal reactor operations can be obtained in a matter of weeks or months using ATR's high neutron flux capability.

A portion of the evaluation if Attila for use at the ATR and INEEL is the ability to visualize it's results. The following section presents the successful visualization of Attila using VisIt.

6 Attila and VisIt

Attila currently provides visualization support via the commercially available Tecplot application. It was our desire to provide visualization through standard tools used at the INEEL. Since VisIt was already in use, we examined what it would take to migrate the Attila output data into VisIt.

The first choice involves either modifying Attila to output a file format currently support by VisIt or creating a conversion utility to migrate current Attila output formats into a format supported by VisIt. The first option is not desirable since Attila is supported outside the INEEL and managing changing source files across multiple locations is not desirable. The second option is the more desirable of these two.

A third option exists, based on the component architecture of VisIt: creation of a VisIt database component based on current Attila output files. Since Attila currently supports visualization by Tecplot, we examined the output files utilized by Tecplot. These files were found to have all mesh definition information and scalar data values required by VisIt. It was decided a new database component would be created, allowing VisIt to utilize existing file formats currently produced by Attila with no modifications.

6.1 Attila Database Component

Creation of the Attila database component required creation of a parsing module for the Attila output file, the Tecplot file in this case. Once we were able to parse the file, all that remained was providing the necessary methods required by the VisIt plugin technology to provide the necessary data to VisIt in the appropriate format.

The VisIt plugin technology allows VisIt to query for available meshes and variables so they may be presented to the VisIt user. Once the VisIt user has indicated the mesh(es) and/or variable(s) to utilize, VisIt is able to retrieve the appropriate data and present it to the plot generation modules.

The Attila mesh information was segmented into smaller pieces rather than providing a single, large mesh. This segmentation fits into VisIt's concept of domains. Each mesh segment is mapped into a VisIt domain. VisIt allows individual domains to be enabled or disabled at any time by the user. This allows the user to easily filter out the domains that are not of interest at any point without having to put together complex operators to filter out the unnecessary data. Domains are also used to determine individual compute tasks in parallel operation. Figures 5 and 6 present views of the full Attila model of the ATR, using a pseudocolor plot and neutron flux variable, and a view with some of the domains removed. Notice in Figure 5 how the full model hides all internal information. Figure 6 has had a portion of the model, specifically the "water" and "reflect" domains.

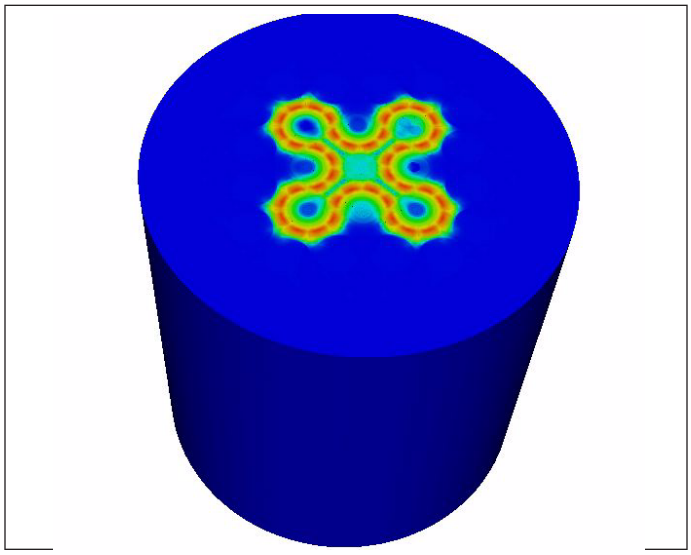

Figure 5 Full Attila Model

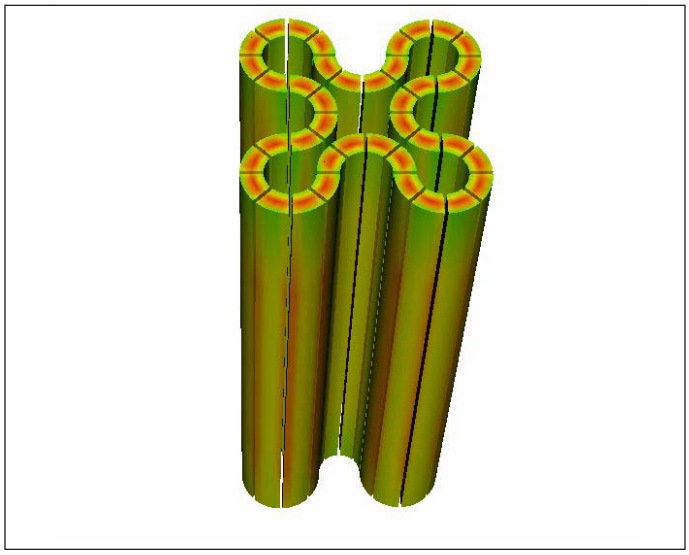

Figure 7 Attila Model of ATR Core

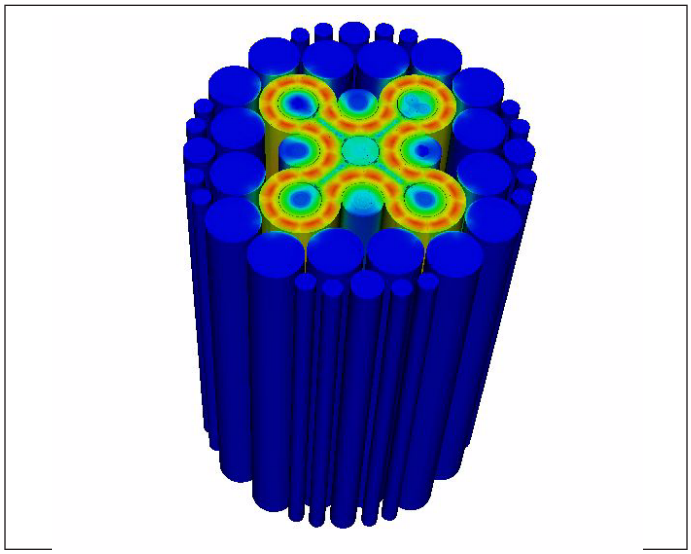

Figure 6 Attila Model With Domains Disabled

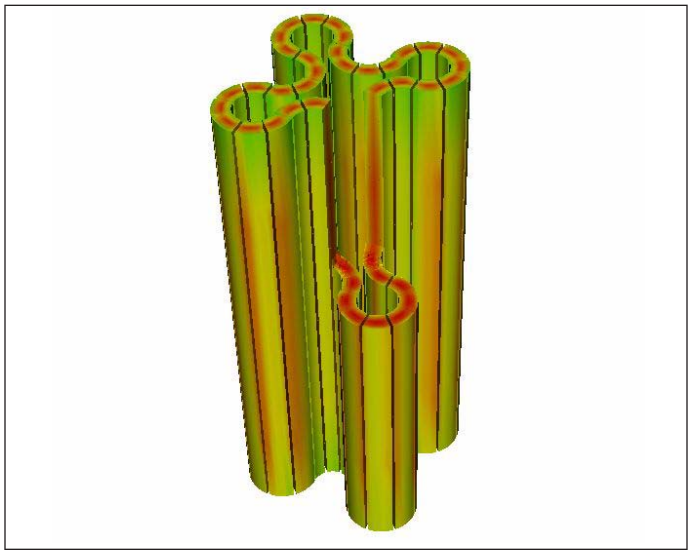

Figure 8 Attila Model of ATR Core with Clip Operator

What remains are the domains that represent the serpentine core of the ATR, several locations where various experiments may be placed, and other various support components.

To view only the core of the ATR, all non-core domains are removed as shown in Figure 7. At this point many of the different operators can be applied to the core to view the neutron flux using different techniques.

Figure 8 presents a plot of the core using the clip operator with three planes to allow viewing an inner portion of the core. The locations of these planes is arbitrary and under complete control of the user to focus in on the necessary portion of the model.

More detailed analysis can be performed by taking a 2D slice of the database in question and retrieving a portion of the resulting database and applying it to a X-Y plot. Figures 9 and 10 presents this concept. The three lower experiment locations had a slice operator applied to them and projected to a 2D view (Figure 9). Once entering a 2D view, Lineout mode can be entered where a line may be drawn to indicate where the data slice is to be taken. This data is then applied to a simple X-Y plot which is displayed in another window (Figure 10). The line drawn in the 2D display is displayed in the same color as the X-Y plot to assist the user in identifying where the data actually came from. Multiple X-Y plots may be created within a single window.

6.2 Platform Availability

Visit supports multiple compute platforms from a single source distribution. Standard configure scripts are used to configure VisIt on Unix based platforms, while specialized Makefiles are provided for Microsoft Windows based platforms.

Our position is that we need to provide the necessary tools on the platforms where they are required. We do not want to force

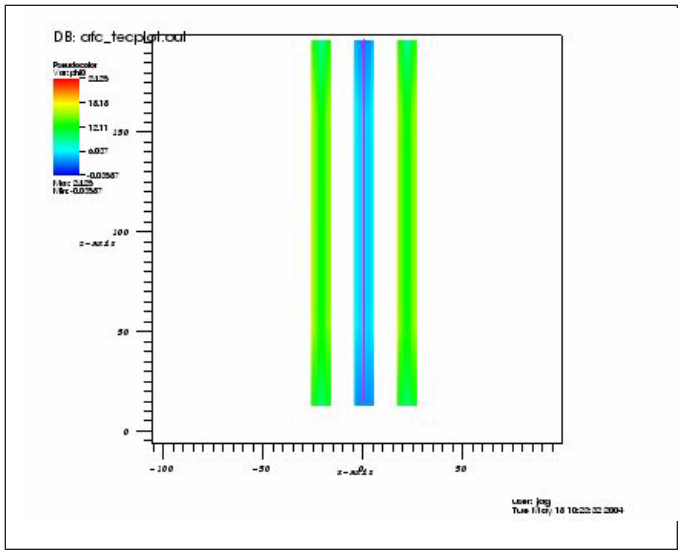

Figure 9 2D Slice In Lineout Mode

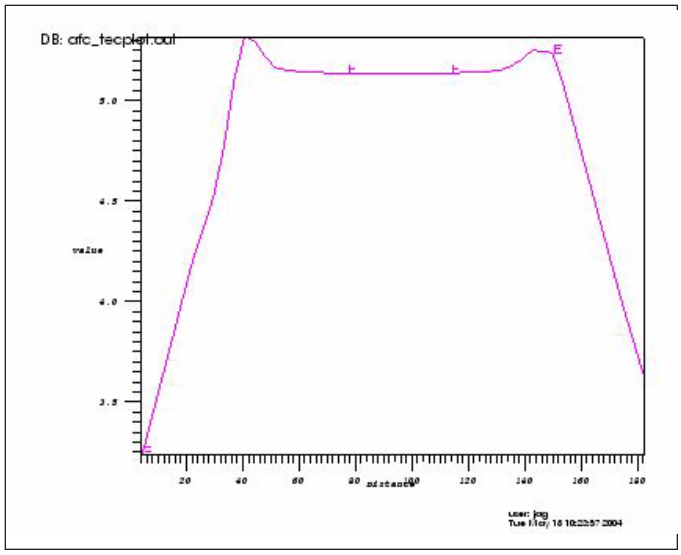

Figure 10 X-Y Plot From Lineout Mode

users to work on an environment that is not compatible with that where their model exists. We have currently implemented VisIt on the following platforms at the INEEL: Sun, SGI, Linux, and MS Windows.

Implementation of Visit in our environments did not present any difficult events. The primary problems we encountered were in the MS Windows environments. See the following section for a discussion on the performance bottleneck on the Windows platform.

6.3 Performance

Performance is always a concern when working with scientific computing. Visualization is no exception. As scientific models get larger, the workload inherited by the visualization tools increases as well.

VisIt addresses the performance issue through its basic architecture. It is component based, allowing different components to be distributed across multiple machines. The compute engine may be run serially on a singe processor or in parallel on potentially thousands of processors, with final rendering performed on the local machine.

Removal of unnecessary domains, will enhance performance since they are removed before the compute engine component actually sees the data.

While our current Attila model does not adequately test the VisIt performance capabilities, we have seen documented evidence of easily handling models with 13.8 million elements. although this level of elements takes on the order of 275 seconds, through the ability to generate in parallel, this time drops down to approximately 25 seconds using 32 processors.

The Attila model presented in this paper consists of 398,775 nodes and 1,361,682 tetrahedral elements. While size of this model does not effectively demonstrate the ability of VisIt to handle large tera-scale models, it does demonstrate it's ability to handle smaller problems and smaller hardware architectures.

A large amount of visualization by VisIt will requires no specialized hardware or software. Images for this paper were generated on a Dell Inspiron 8000 laptop with a P4 800Mhz processor with standard integrated graphics. We have several users utilizing VisIt on their desktop Linux workstations, both single and multiprocessor environments, with adequate performance.

In our Attila database implementation, we noticed severe performance problems while parsing the Attila plot file. The bottleneck was traced to the STL containers being used.

Recognizing that the original STL implementation provided by MS Visual C++ is not the most efficient, we set about installing and integrating the STLport [5] STL implementation. This required changing some of the VisIt source files to remove all non-standard STL include references, primarily the iostream related includes. This was required due to the fact that STLport provides it's iostream implementation in shared libraries rather than include files.

Once this was all straightened out, we noticed a significant improvement in performance. In fact, the MS Windows implementation now runs faster that a Linux implementation on a mcahine with a faster clock. The original MS Windows implementation required approximately 1.5 minutes to read the Attila input file (94,435,979 bytes) on a 800 Mhz class machine while the Linux implementation only required 15 seconds on a 2 Ghz class machine. Following implementation of the STLport STL libraries, the MS Windows implementation now only takes 10 seconds on the 800 Mhz class machine.

It is apparent that the STL library implementation in the Windows environment has a significant impact on a database reader implementation using STL containers and iostreams. Detailed performance differences of the rendering engine of VisIt were not done before and after the STLport implementation. Based on the performance we see in the database reader implementation, it leads us to believe that the rendering engine

would also see significant performance improvements in the areas where STL containers or iostreams were utilized.

7 Conclusion

We have found that the VisIt visualization tool is quite adequate, both in performance and functionality, for our current and expected scientific visualization problems. It provides extensive functionality in it's standard plot types and operators and provides the ability to extend the existing set of plot types, operators, and supported databases via it's plugin capabilities.

We are able to utilize VisIt on a wide array of hardware platforms and are not currently concerned with performance implications. As the models become larger, we will utilize VisIt's parallel capabilities in a cluster environment as necessary.

While the user documentation is adequate, the documentation concerned with the extension of the plot, operator, and database capabilities is minimal and not yet available via the VisIt website. While this is true, we have found very good response via the telephone and e-mail to various software engineers involved with VisIt's design and development and received much information, including sample code, to assist us in our endeavours.

VisIt also supports the need for reporting results in formal documents and making "movies" or animations for demonstrations through it's ability to generate output in several different standard graphic file formats.

While other tools may provide similar capabilities, VisIt was found to provide the necessary functionality and performance for our current and anticipated needs. It is definitely a generic solution to a generic problem. And, by the way, the price is right.

About The Authors

James A. Galbraith is a Software Engineer in the HPC/Visualization group at the INEEL. He is currently involved in migrating various applications to utilize VisIt for visualization of scientific data and addressing data migration issues. He may be reached at galbja@inel.gov. L. Eric Greenwade is the group lead for the HPC/Visualization group at the INEEL. He may be reached at leg@inel.gov.

References

[1] Attila, Los Alamos National Laboratory, www.lanl.gov/attila.

[2] Attila, Radion Technologies, www.radiative.com.

[3] AVS, Advanced Visual Systems, Inc. www.avs.com.

[4] ParaView, KitWare, Inc. www.paraview.org.

[5] STLport, STLport Consulting, www.stlport.org.

[6] Tecplot, Tecplot, Inc., www.tecplot.org.

[7] VisIt, Lawrence Livermore National Laboratory, www.llnl/visit.

[8] VTK, KitWare, Inc., www.vtk.org.